CS 225
Data Structures

Feb. 21 – Binary Search Tree
Wade Fagen-Ulmschneider
Interactive Lecture Questions

- Ask Questions: Ask in-lecture questions using this Google Form!
- Detailed Answers After Lecture: If we didn't get to answer your question in lecture, we provide detailed answers to common questions here.
- You must be logged in with an active University of Illinois email address to access the form. If you don't see the form, please be asked to log in.

Lecture Videos

- Recorded on echo360.org, log in with your University of Illinois email address.

Schedule

- **Monday**
  - January 15
    - MLK Day
  - January 22
    - Memory
      - slides | handout | pointers.pdf | code | TA Notes

- **Tuesday**
  - January 23
    - Slides 2 | Slides 3 | handout | Binky Pointer Fun | code | LTA Notes

- **Wednesday**
  - January 24
    - Slides 4 | handout | code | LTA Notes

- **Thursday**
  - January 25
    - Slides 5 | handout | code | LTA Notes
Traversals vs. Search

Traversals vs. Search:
- **Traversal** visits every node in the tree exactly once.
- **Search** finds one element in the tree.
Search: Breadth First vs. Depth First

**Strategy:** Breadth First Search (BFS) / Traversal

**Strategy:** Depth First Search (DFS) / Traversal
Running Times on a Binary Tree
Dictionary ADT

Data is often organized into key/value pairs:

UIN  ➔ Advising Record
Course Number  ➔ Lecture/Lab Schedule
Node  ➔ Incident Edges
Flight Number  ➔ Arrival Information
URL  ➔ HTML Page
...

#ifndef DICTIONARY_H
#define DICTIONARY_H

class Dictionary {
public:

private:

};

#endif
Binary Tree as a Search Structure
Binary ______________ Tree (BST)

A BST is a binary tree $T$ such that:
#ifndef DICTIONARY_H
#define DICTIONARY_H

template <class K, class V>
class BST {

public:
    BST();
    void insert(const K key, V value);
    V remove(const K & key);
    V find(const K & key) const;
    TreeIterator traverse() const;

private:
    struct TreeNode {
        TreeNode *left, *right;
        K & key;
        V & value;
        TreeNode(K & k, V & v) : key(k), value(v), left(NULL),
            right(NULL) { }
    };

};

#endif
template<typename K, typename V>

TreeNode *root;

find(TreeNode *root, const K &key) const {
    // Tree data structure
    // root node with values
    // 38: 10, 13, 51
    // 13: 10, 12
    // 25: 40, 37
    // 51: 84, 66, 89, 95

    // Implementation of find function
    if (root == nullptr || root->key == key) return root;
    if (key < root->key) return find(root->left, key);
    return find(root->right, key);
}
template<typename K, typename V>

insert(TreeNode *& root, const K & key) {

}
template<typename K, typename V>

TreeNode* _remove(TreeNode* & root, const K & key) {

}
remove(40);
remove(25);
remove(10);
remove(13);